VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Absolvování individuální odborné praxe

# Individual Professional Practice in the Company

2014

Karel Antošík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Karel Antošík**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: KVADOS, a.s.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **RNDr. Eliška Ochodková, Ph.D.**

Konzultant bakalářské práce: Ing. Radek Garzina, Ph.D.

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych tímto poděkoval svému konzultantovi na odborné praxi, panu Ing. Radku Garzinovi, Ph.D., za pečlivé vedení během praxe, profesionální dohled a cenné rady a zkušenosti, které mi byly předány. Mé poděkování patří též mé vedoucí bakalářské práce RNDr. Elišce Ochodkové, Ph.D., za věcné připomínky a vstřícnost při konzultacích k samotnému vypracování.

## Abstrakt

Tato bakalářská práce se zabývá průběhem odborné praxe ve firmě KVADOS, a.s. První část se věnuje popisu firmy KVADOS, a.s. a jejich produktů, dále pak pracovnímu zařazení studenta s představením dvou klíčových systémů, se kterými se při odborné praxi setkal. V další sekci jsou uvedeny a popsány úkoly, které byly zadány studentovi během praxe, přičemž u každého z úkolů je poté vysvětleno, jakým způsobem student problém řešil a jaké technologické prostředky k tomu použil. Předposlední část zahrnuje souhrn znalostí, a to jak těch, které student během praxe uplatnil, tak těch, které postrádal a vzápětí nabyl. Na závěr se práce dostane k celkovému hodnocení odborné praxe a jejich výsledků.

**Klíčová slova:** VENTUS, TFS, .NET, ADO.NET, C#, T-SQL, SQL Server, ASP.NET, XML

## Abstract

This thesis deals with professional practice in the company KVADOS, a.s. The first part is devoted to the description of the company KVADOS, a.s. and its products, as well as student's grade during the professional practice with the introduction of the two key systems which the student on the practice met with. In the next section are said and described tasks that were given to student during the practice with explanation of how the student solved problems and what technological means were used. Penultimate section includes a summary of knowledge as those which the student applied during the practice, as those which were missing and then learned. In the end of thesis is rating of the professional practice and its results.

**Keywords:** VENTUS, TFS, .NET, ADO.NET, C#, T-SQL, SQL Server, ASP.NET, XML

## Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| ADO.NET | – | Microsoft ActiveX Data Objects .NET |
| ASP | – | Active Server Pages |
| IS | – | Informační systém |
| LINQ | – | Language Integrated Query |
| MS | – | Microsoft |
| ORM | – | Objektově-relační mapování |
| SMTP | – | Simple Mail Transfer Protocol |
| SQL | – | Structured Query Language |
| TFS | – | Team Foundation Server |
| T-SQL | – | Transact-SQL |
| XML | – | Extensible Markup Language |

# Obsah

# Seznam obrázků

# Seznam výpisů zdrojového kódu

# 1 Úvod

Informační systémy denně ulehčují práci milionům lidem, ať už se jedná o velké dopravní projekty nebo malé firemní softwary. V dnešní době si stěží dovedeme představit armádu zaměstnanců, hledající data uprostřed hromady papírů. V současnosti řeší vyhodnocení a obecně zpracování dat čím dál tím častěji automatizované, převážně softwarové informační systémy, které nám na jedno kliknutí vychrlí vše žádané.

V této bakalářské práci popisuji svůj průběh odborné praxe ve firmě KVADOS, a.s. Ta byla zaměřena na implementaci rozhraní mezi IS VENTUS (věnuji se v kapitole 2.3.1) a s "work itemy"nástroje Team Foundation Server (dále jen TFS) společnosti Microsoft (věnuji se v kapitole 2.3.2). Obsahově je práce rozdělena na několik částí, kde v první z nich představuji firmu, dva klíčové produkty, se kterými jsem se při implementaci setkal a pracovní zařazení v rámci odborné praxe. Dále se v práci věnuji popisu zadaných úkolů a jejich následnému řešení. Nakonec se pokusím zhodnotit dosažené výsledky, své znalosti, ať už ty, které jsem při praxi získal, tak ty, které jsem nabyl již během studia a v praxi je uplatnil.

# 2 Představení firmy a pracovního zařazení

## 2.1 KVADOS, a.s.

[1]Společnost KVADOS, a.s, jejíž název znamená KVAlitní DOdavatelské Služby, je česká firma s centrem v Ostravě, zaměřená na vývoj a dodavatelství vlastních softwarových řešení, a to zejména pro klienty ze segmentu obchodu a služeb. Na trhu působí od roku 1992 a v současné době dodává své produkty do 12 evropských zemí. Její tým odborníků čítá kolem 130 lidí a stále se rozrůstá.

Mezi firemní produkty patří mimo jiné mobilní informační systém myAVIS™, který posbíral řadu ocenění Microsoft Awards. Dále pak výčtem myFABER™, myWORK™, mySTOCK™, myTEAM™, myMACHINE™, myCASH™ a myDATACENTER™.

## 2.2 Pracovní zařazení

V rámci odborné praxe jsem byl zařazen do oddělení vývoje .NET aplikací, kde jsem pracoval na projektu rozhraní, synchronizující úkoly programátorů z firemního produktu VENTUS® a work itemy nástroje TFS. Na tomto projektu jsem pracoval s jedním studentem z VŠB-TUO, který byl ve firmě také na odborné praxi. Projekt byl proto rozdělen na dvě části, kdy každý z nás se věnoval jedné z nich. Kolega se věnoval části mezi synchronizační databází a TFS, zatímco já jsem pracoval na rozhraní mezi Ventem a synchronizační databází.

Náplň mé práce během odborné praxe byla především implementace jednotlivých částí systému v programovacích jazycích C# a T-SQL. Zejména se jednalo o zařízení komunikace aplikace s firemním MS SQL Serverem a dále o komunikaci mezi aplikací a synchronizační databází. Nejednalo se však pouze o tok dat, ale i o řešení chybových stavů, databázových konfliktů nebo parametrizaci pomocí XML konfiguračního souboru.

## 2.3 Popis současného stavu

Momentálně je v KVADOSu situace taková, že existuje firemní systém VENTUS, který je určený pro personalistiku a zároveň TFS jako produkt, jenž má roli "source control". A proto výsledkem projektu, na kterém jsem během své odborné praxe pracoval, by mělo být umožnění přímo z Venta, za pomocí synchronizace mezi jednotlivými systémy, řídit proces vývoje.

### 2.3.1 IS VENTUS®

[2]VENTUS® je informační systém vyvinutý společností KVADOS, a.s., který je určen středním a větším firmám. Jeho úlohou je zefektivnění činností a firemních procesů, speciálně v oblasti logistiky a distribuce zboží. Dle firemného webu je tento systém vyvinut

[1]V tomto textu jsou použity informace z webu www.kvados.cz/Content/O-spolecnosti
[2]V tomto textu jsou použity informace z webu www.kvados.cz/Content/VENTUS-Software

na technologii softwarových agentů, kteří umožňují na základě rozsáhlých variant konfigurace nastavit chování a především posloupnost procesů dle konkrétních požadavků klienta.

### 2.3.2 TFS

TFS je nástroj vyvinutý společností Microsoft určený pro řízení procesu vývoje aplikací. Je primárně navržen a podporován jako rozšíření vývojových nástrojů MS Visual Studio a Eclipse.

Hlavním pojmem, se kterým se u TFS setkáme je takzvaný "source control". Je to nástroj pro sdílení kódu a kontrolu vývojových verzí projektů. Sekundárně je TFS určen k systému řízení vývoje softwaru, kde se setkáme s důležitým prvkem, se kterým jsem se při své odborné praxi pracoval, a to takzvaným "work itemem". Pod ním si můžeme představit jakýkoliv problém, úkol či cokoliv jiného, co se týče vývoje softwaru. Mezi základní atributy patří typ, název, datum požadovaného dokončení a další, ale jelikož jsou dobře rozšiřitelné, můžeme si vytvořit, díky vestavěnému editoru, vlastní šablonu úkolu, který bude mít námi definované atributy.

Další vlastností těchto pracovních položek je možnost vytvořit mezi nimi vztahy, a tím vytvořit logický provázaný strom [1].

# 3 Zadání úkolů během praxe

## 3.1 Analýza a mapování

Mým prvním úkolem bylo seznámit se s datovým modelem systému VENTUS®, zanalyzovat ho a následně potřebnou část namapovat. Jednalo se o systém úkolů se vztahy na rodičovské projekty, na přiřazené, či vytvářející uživatele a další atributy, které by byly v budoucnu nezbytné jako například časová náročnost nebo data vytvoření, či požadovaného dokončení apod. Vzhledem k rozsáhlé datové struktuře, kdy například tabulka s úkoly měla 138 atributů a na ní se teprve navazovaly tabulky s kategorií, nebo typem úkolu, jsem strávil nad řešením tohoto úkolu několik dnů s tím, že výsledek nebyl zdaleka konečný a při řešení dalších zadání se tato objektově relační vrstva rozšiřovala či zužovala dle systémových požadavků.

## 3.2 Vytvoření servisní vrstvy

Cílem tohoto úkolu bylo vytvořit jednoduché komunikační rozhraní aplikace s testovací instancí databáze MS SQL Serveru. Zejména se jednalo o metody selektující data dle určitých parametrů, dále pak funkce pracující s úkoly a to například funkce na uzavření úkolu, nebo jeho následovné znovuotevření. Parametry pro filtrování byly především primární klíče entit, hodnoty číselníků, různá data a jejich četné kombinace.

Specifické pro toto zadání bylo, že funkce, které vracely záznamy z tabulky, pracovaly s návrhovým vzorem Továrna. Dále bylo po mně požadováno, aby každá funkce ze servisní vrstvy komunikovala s databází pomocí uložené procedury.

Časová náročnost tohoto úkolu byla z důvodů implementace jak aplikačních funkcí, tak uložených procedur přibližně patnáct dní. Do tohoto časového horizontu spadá i následné testování funkčnosti a odladění chyb.

## 3.3 TFS kooperační databáze

Důležitou součástí systému byl prostředník mezi Ventem a jeho úkoly a TFS databází "work itemů". Tímto prostředníkem se stala kooperační databáze, uchovávající si data přiřazených úkolů, jejich návazností na kategorii, typ a stav. Důležitou součástí pak byla vazba "work itemu"na TFS projekt a časová známka v podobě data vložení záznamu, či jeho poslední aktualizace. Toto datum v dalších úkolech sloužilo k sledování změn a následné možnosti synchronizace. Do její datové struktury se dále řadili uživatelé a v návaznosti jejich vztahy na úkoly. Podstatou této databáze pak bylo uchovávní změn, které se pak propagovaly ven ať už směrem do systému VENTUS, nebo naopak do TFS.

Mým úkolem zde bylo přelévat data právě z firemního produktu do kooperační databáze. Kritické zde bylo ošetřování datové struktury při vkládání, nebo aktualizaci dat. V této fázi už bylo nutné přemýšlet i nad fungováním synchronizace a případným rozšířením nebo změněním datové struktury. Nejen proto, ale i kvůli simulaci přelévání dat, testování funkčnosti uložených procedur a ošetřování správného vkládání dat do entit byla časová náročnost tohoto úkolu kolem dalších patnácti dnů.

## 3.4 ASP.NET portál

Mým dalším úkolem na odborné praxi bylo vytvořit ASP.NET portál, který by filtroval a zobrazoval úkoly dle zadaných parametrů. Mezi tyto parametry patřily kódy, a to rodičovského projektu, přiřazeného uživatele a vytvářejícího uživatele. Dalšími nutnými prvky byly datum vytvoření a hodnota číselníku stavů úkolů. Dále po mně bylo požadováno, aby tento portál uměl stránkovat výsledky hledání a dynamicky měnit počet záznamů na jedné stránce.

Tento portál vznikl z několika důvodů. Především se jednalo o možnost specifikovat k daným úkolům jednotlivé TFS projekty, jelikož automatická vazba nemohla vzniknout. Dále pak vizualizace listu vyfiltrovaných úkolů a tím umožnění sledování stavu jejich aktuálního řešení. Další z mnoha výhod a cílů portálu bylo otestování efektivity a rychlosti selektovaných dat.

Vzhledem k řešení spousty problémů, se kterými jsem měl malé zkušenosti se časová náročnost tohoto úkolu pohybovala kolem deseti dnů.

## 3.5 Synchronizační Service

Jedním z posledních úkolů, ale pravděpodobně tím nejpodstatnějším, bylo vytvoření aplikace, která by byla schopna při spuštění synchronizovat data. Toto zadání jsem dostal za úkol řešit pomocí konzolové aplikace psané v jazyku C#. Důležitým prvkem aplikace bylo načítání parametrů z XML konfiguračního souboru, ve kterém byla mimojiné data pro filtrování (uživatelé a číselník stavů). Dále se v tomto souboru vyskytovaly "connection stringy"k jednotlivým databázím, ať už systému VENTUS, nebo kooperační instanci k TFS. Nakonec soubor obsahoval definici SMTP serveru a výčet mailových adres, na které se při chybě synchronizace měla poslat varovná zpráva.

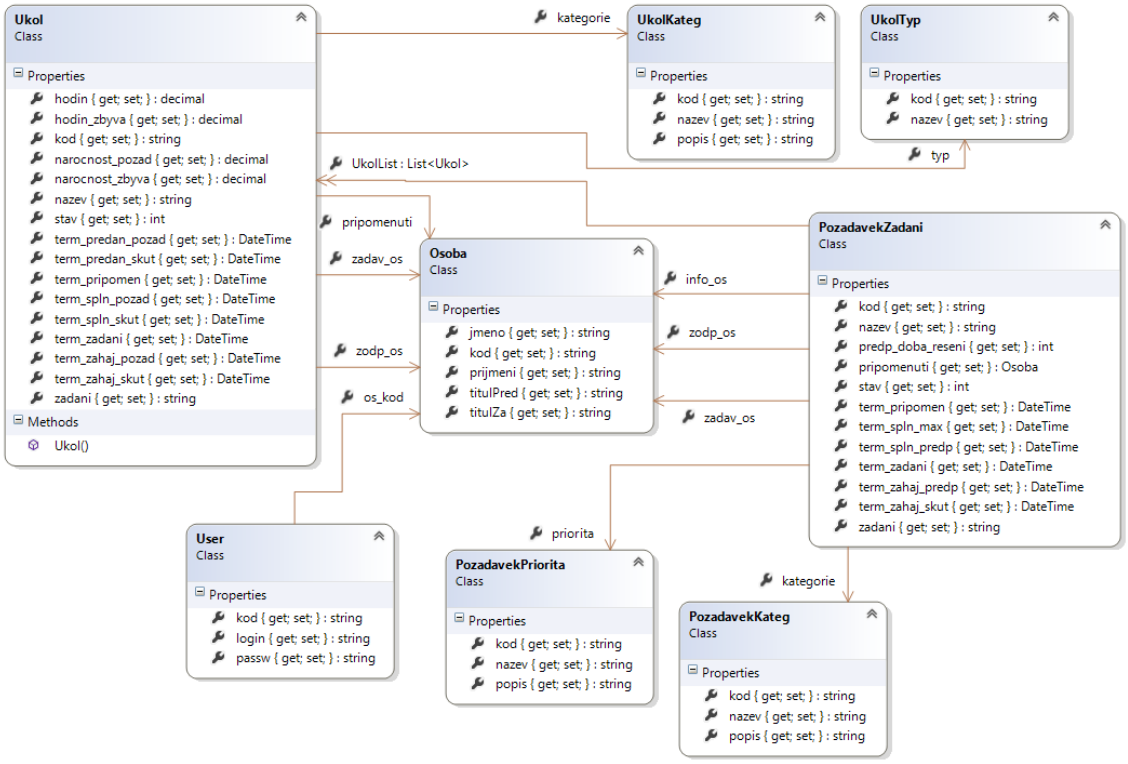
Tato aplikace dále musela umět získat datum poslední aktualizace a dle něj vybrat potřebná data k aktualizaci a případnému vložení. Jelikož jsem při tomto úkolu musel upravit v předešlých úkolech implementované metody a zároveň nové funkce, například pro načítání dat z XML a jiné, přidat, časový horizont řešení se pohyboval kolem dalších deseti dnů.

Výsledná aplikace se měla spouštět v časovém intervalu pěti minut, případně manuálně i dříve. O to se měl starat plánovač úloh operačního systému Windows. K tomuto řešení ale nezbyl během mé odborné praxe čas, a tak se mé úkoly staly čistě implementační a testovací záležitostí.

# 4 Řešení zadaných úkolů

## 4.1 Analýza a mapování

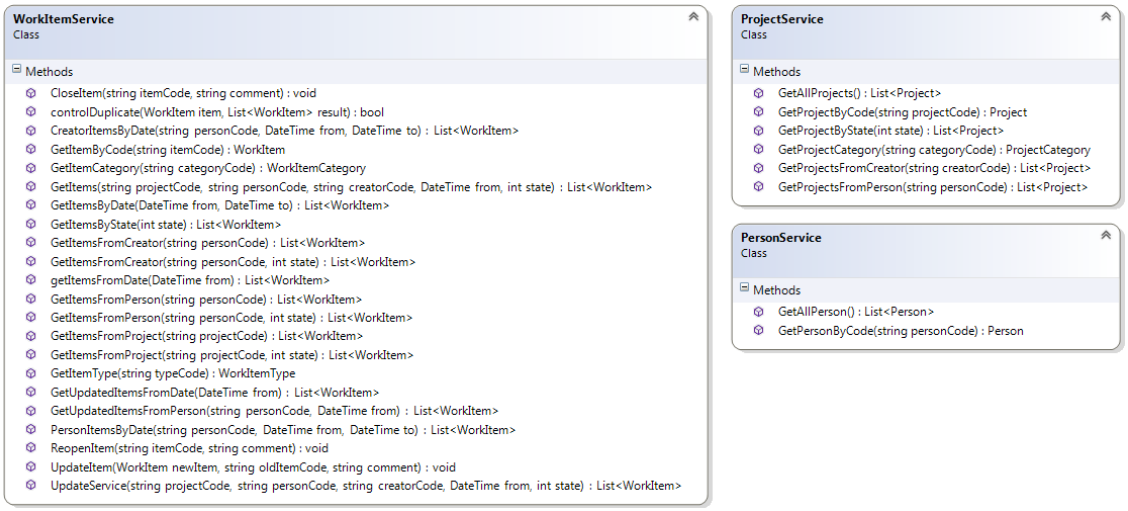
Při řešení tohoto úkolu jsem se nesetkal s žádnými nepřekonatelnými překážkami. Jak jsem již uvedl v zadání, kvůli rozsáhlé datové struktuře systému, jsem strávil nad tímto úkolem několik dní, kdy jsem provedl spoustu selectů nad databází a hledal v nich jednak potřebné atributy, jednak cizí klíče jako reference na jiné entity. Jednotlivé výsledky a poznatky provázanosti entit jsem si následně zaznamenával na papír a posléze, když už jsem hrubou verzi objektově-relačního modelu měl, je přepsal v programovacím jazyku C# do tříd a jejich properties. Všechny tyto třídy byly součástí samostatné assembly pro maximální možnou flexibilnost v budoucím budování synchronizačního systému. Na následujícím obrázku je zachycen diagram hrubé verze ORM vrstvy. V této fázi byly všechny proměnné a třídy pojmenovány česky podle databázových entit a jejich atributů. Během následujícího vývoje systému se všechny názvy přeložily do anglického jazyka a byl jim udán jednotný tvar jak pro úkoly z Venta, tak pro TFS "work itemy"z důvodu dodržení zavedených programovacích standardů. To znamená, že například třída Ukol byla následně po přejmenování třídou WokItem, třída PozadavekZadani se názvem změnila na Project a podobně.




Obrázek 1: Prvotní návrh a implementace objektově-relační vrstvy

## 4.2 Vytvoření servisní vrstvy

Pro účely komunikace s databází jsem si vytvořil servisní třídu, které se v konstruktoru předal parametr "connection string", což je řetězec popisující připojení. Tato třída měla implementované metody jednak pro provádění selektovacích příkazů, tak i pro transakce a příkazy nevracející data. K řešení jsem použil funcionality technologie ADO.NET, která představuje množinu tříd nabízejících služby pro přístup k datům a tvorbu databázových aplikací [2]. Následující diagram zachycuje výčet servisních funkcí, které byly součástí mé prvotní implementace. Tendence počtu metod byla během následujících úkolů spíše klesající, a to z důvodu jejich nevyužití.




Obrázek 2: Diagram servisních metod

Pro každou databázovou entitu, se kterou jsem pracoval, jsem si vytvořil vlastní servisní třídu a v každé z nich jsem následně implementoval jednotlivé metody. Svou prvotní verzi procedur jsem následně přeprogramoval tak, aby objekty byly vytvářeny pomocí návrhového vzoru Továrna. Pro tyto "továrny"jsem si vytvořil vlastní modul, ve které byly třídy uchované. Každé takové továrně jsem naimplementoval funkci přijímající jako argument SqlDataReader, která tento reader rozparsovala do návratového objektu. Součástí parsování bylo rovněž ošetření nulových hodnot jednotlivých sloupců entity.

Co se týče samotných servisních metod, většina z nich byla naprogramována stejným stylem, jako je uvedeno ve výpisu kódu 1. Funkce nejdříve otevře databázové spojení, následně vytvoří příkaz typu uložená procedura, kterému se přidá potřebný parametr, v tomto případě kód rodičovského projektu. Dále se zavolá pomocná procedura Select mnou vytvořené komunikační třídy s parametrem příkazu. Ta vrátí SqlDataReader s požadovanými daty, který je posléze v cyklu rozparsován do objektu WorkItem a ten je přidán do generického listu. Nakonec se reader i databázové spojení zavřou a list s požadovanými daty funkce vrátí.

Dalším typem implementovaných metod byly ty, které aktualizovaly stavy úkolů systému Ventus. Mezi ně patřily například funkce, které uzavíraly nebo znovu otevíraly úkoly. Tato aktualizace stavu proběhla podobně jako výše uvedená a popsaná metoda s tím rozdílem, že místo metody Select se zavolala funkce ExecuteNonQuery, které se předal příkaz typu uložená procedura.

Oba moduly, jak "továrny", tak modul se servisními metodami byly stejně jako ORM v kapitole 4.1 součástí samostatné assembly pro usnadnění budoucí manipulace.

## 4.3 TFS kooperační databáze

Jako komunikační prostředek s kooperační databází jsem si v servisním modulu vytvořil novou třídu. Tato třída obsahovala funkce pro vkládání a aktualizaci dat, ale například také metodu, která vrátila datum poslední aktualizace. Ve výpisu kódu 2 je ukázána funkce aktualizující data v kooperační databázi. Toto datum bylo do budoucna velmi potřebné z důvodu zjišťování nových změn. Veškerá komunikace opět probíhala pomocí uložených procedur a z důvodu možné kolize se veškeré změny prováděly v transakcích s možností funkce rollback.

Aplikace sama měla možnost volat vkládání a aktualizaci pouze na objekt WorkItem a o vše ostatní se dále starala zavolaná uložená procedura. Té se nejprve předaly parametry nutné k úspěšnému naplnění datové struktury. V provázanosti entity na jiné tabulky se muselo ošetřit i vložení ostatních dat, a to především nových uživatelů, kategorií a typů úkolů a dále pak data pro relační tabulku, která nám zajišťovala m:n vazbu tabulky Workitem a User. To se provedlo tak, že uvnitř transakce se postupně zjišťovala existence řádku v dané tabulce a následně, pokud záznam neexistoval, zavolala se jiná příslušná uložená procedura, která vložila nový řádek. Pokud došlo během vkládání někde ke konfliktu, procedura pak byla schopná pomocí rollbacku vrátit všechny změny a nahlásit aplikaci pomocí návratové proměnné chybu synchronizace.

Po implementaci došlo k testování funkčnosti přelévání dat. Díky servisní vrstvě 4.2 jsem si byl schopen vytáhnout potřebná data a následně pomocí metod výše popsaných je uložit do kooperační databáze. Pro účely testování jsem použil konzolovou aplikaci, ve které jsem nejdříve odzkoušel úspěšené vkládání a aktualizaci dat. Následně jsem záměrně vyvolával chyby synchronizace tím, že jsem posílal nesprávná data a otestoval tím rollback a ohlášení aplikaci chybových stavů..

## 4.4 ASP.NET portál

Jak jsem již naznačil v zadání toho úkolu, podstatou tohoto portálu bylo otestovat rychlost selekce dat. Jako první jsem si ve svém řešení založil nový ASP.NET projekt, do kterého jsem si na základní stránku přidal potřebné kontroly pro výběr parametrů a zobrazování výsledků filtrování. Co se týče parametrů, pro všechny parametry kromě vybírání data jsem použil ComboBoxy a pro výběr data základní webovou kontrolu Calendar. Na zobrazování dat jsem použil GridView, pro jeho možnost stránkování a dynamickému měnění počtu záznamů na stránku.

Když jsem měl uživatelské rozhraní hotové, začal jsem pracovat na vytváření a přiřazování datových zdrojů jednotlivým komponentám. Jelikož všechny tyto zdroje měly být objektové, používal jsem především již naimplementované metody ze servisní vrstvy, které vracely objekty v podobě generických listů. Tyto funkce jsem proto musel poupravit tak, aby list, který vracejí, byl abecedně seřazen. Jediný datový zdroj, který jsem musel naprogramovat byl číselník stavů, který jsem pro prvotní účely vytvořil jako list s napevno danými hodnotami.

Dalším krokem v plnění úkolu bylo vyfiltrování dat dle parametrů popsaných výše a jejich zobrazení v předpřipraveném GridView. Nejdříve jsem pro toto využil metody s příslušnými parametry ze servisní vrstvy, které vracely generické listy s objekty typu WorkItem. Následně jsem provedl průnik těchto listů pomocí integrovaného jazyka LINQ a výsledek zobrazil v tabulce. Toto řešení sice fungovalo, ale bylo moc pomalé, a tak jsem začal hledat jinou alternativu, která by selekci dat urychlila a tím zeefektivnila aplikaci. Rozhodl jsem se, že rychlost vyřeším na úrovni MS SQL Serveru, a to uloženou procedurou s možností dynamického parametrování selektujícího příkazu.

Tato procedura (výpis kódu 3) měla zadaný základní příkaz, ke kterému se dle hodnot zadaných parametrů přidávaly další argumenty. Vyhodnocování parametrů probíhalo tak, že ty, jejichž hodnota byla typu textového řetězce, se porovnávaly s prázdným textem a pokud řetězec prázdný nebyl, argument se přidal. Čísleník stavů se porovnával s nulou a nakonec datum se vyhodnotilo dle toho, jestli se rovnalo datu 1. 1. 1753, což je datum, které je aplikačně vráceno při vytvoření nové proměnné typu DateTime. Při řešení a implementaci této procedury jsem čerpal z následující reference [3].

## 4.5 Synchronizační Service

Plnění tohoto úkolu jsem začal založením nového projektu ve svém řešení. Tento projekt byl typu konzolové aplikace, ke které jsem přidal potřebné reference na servisní i ORM vrstvu. Nejprve jsem se pustil do řešení načítání parametrů z konfiguračního XML souboru.

Jak jsem uvedl v zadání, konfigurační soubor měl obsahovat popisy připojení k databázím, dále pak skupinu emailových adres a definici SMTP serveru. To nejdůležitější, co bylo uloženo v tomto souboru, byly parametry pro synchronizaci jednotlivých databází. Přesněji šlo o skupinu uživatelů, jejichž úkoly se dle jejich kódu měly vyfiltrovat a číselník stavů jednotlivých úkolů. Nejdříve jsem si vytvořil třídu ConfigHelper, která ve své konečné fázi obsahovala statické metody vracející požadovaná data. Co se týče skupin osob, emailových adres a číselníku stavů, rozhodl jsem se pro načtení dat do paměti využít deserializaci částí XML souboru. Pro deserializaci jsem si vytvořil jednotlivé třídy, které měly list návratových objektů a statickou metodu, která potřebná data načetla ze souboru a deserializovala je do tohoto listu. Jelikož ostatní hodnoty v XML byly unikátní, pro jejich získání jsem pouze vyhledal určitý element a jeho hodnotu načetl do paměti. Po odzkoušení správného načítání dat jsem začal řešit další problém, a to zasílání emailů při chybě synchronizace. Tyto stavy jsem zjišťoval v uložené proceduře, při pokusu o vložení nových "work itemů". Uložená procedura vrátila aplikační metodě výsledek a dle něho se rozpoznalo, zda aktualizace či vložení proběhlo v pořádku, nebo došlo k chybě.

V ukázce výpisu kódu (příloha 4) je implementace statické metody pro zasílání chybových stavů. Této metodě byl předán parametr typu textového řetězce, jehož hodnota obsahovala chybovou hlášku s kódem úkolu, při kterém během synchronizace došlo ke kolizi. V těle metody se nejprve deklaruje instance třídy ConfigHelper popsané výše s parametrem cesty k XML. Dále tuto instanci požádám o první člen listu s SMTP servery a jeho hodnotu uložím do lokální proměnné. Posléze vytvořím zprávu, které nastavím atribut odesílatele získaného z instance serveru. Dalším krokem je přidání emailových adres načtených z XML do kolekce příjemců. Posledními kroky k vytvoření zprávy je nastavení jejího předmětu a těla. Nakonec pro úspěšné zaslání jsem si vytvořil instanci třídy SmtpClient a té jsem nastavil atributy host, zapnutí šifrování a pokud jsou v XML souboru definovány, jsou nastaveny i přihlašovací údaje k SMTP serveru. Posledním krokem bylo zavolání metody Send instance klienta s parametrem zprávy.

Dalším důležitým funkčním prvkem bylo získání správného data poslední aktualizace směrem od Venta do synchronizační databáze. Pro tyto účely byly v datové struktuře obsaženy časové známky a ty se dělily na dva typy. V jedné bylo uloženo datum vložení daného záznamu a v druhé datum poslední manipulace se záznamem. Pro mne byla klíčová časová známka, která zaznamenávala poslední úpravu. Z té jsem si pomocí agregační funkce vyhledal maximum a to vrátil aplikaci. Problém by v té chvíli ale nastal, pokud by záznam byl upraven z opačného směru, tedy z TFS. Z toho důvodu se do datové struktury musel zavést další časový atribut, který hlídal datum poslední úpravy z opačného směru. Tím se zabránilo možnému konflitku aktualizací z obou stran. Po otestování implementované funkce pro získání data jsem se pustil do otestování funkčnosti celé aplikace.

Zde jsem narazil na jediný problém, a to neaktuálnost dat ve vývojové instanci databáze MS SQL Serveru. Z toho důvodu jsem musel datum poslední aktualizace napevno vložit do kódu. Po úspěšném testu synchronizace jsem dále vyzkoušel poslat konfliktní data do komunikačního prostředníka, což mělo zapříčinit zaslání chybových mailů na adresy definované v XML. Když i tato část fungovala správně, aplikace byla hotová.

# 5 Znalosti a dovednosti získané v průběhu studia uplatněné na odborné praxi

Na své odborné praxi jsem využil spoustu znalostí, které jsem získal při dosavadním studiu. Jednalo se především o dovednosti spojené s informačními systémy a programováním v jazyce C#. Tyto znalosti jsem v převážné většině nabyl v předmětech Databázové a informační systémy (DAIS), Vývoj informačních systémů (VIS) a Programovací jazyky II., které byly zaměřeny na jazyk C#.

Co se týče předmětu DAIS, z něho jsem si odnesl především komunikaci s databází pomocí technologie ADO.NET. To znamená selektování dat, vkládání, aktualizaci a mazání záznamů. Dále pak základní dovednosti s implementací jednoduchých ASP.NET aplikací, a to zejména zobrazování požadovaných výsledků a znalost ovládacích prvků.

Paralelně s odbornou praxí jsem studoval předmět VIS, ze kterého jsem si odnesl a využíval znalosti návrhových vzorů pro práci s daty. Převážně to byly vzory Table Data Gateway, Row Data Gateway a další.

Mezi další znalosti, které jsem využil patřilo objektové programování a znalost návrhových vzorů, kdy jsem především využil vzor Továrna.

# 6 Znalosti a dovednosti scházející studentovi v průběhu odborné praxe

Scházejících znalostí během odborné praxe nebylo mnoho, avšak některé se vyskytly. Šlo zejména o znalosti, které jsme probírali až v posledním semestru v předmětu Architektura technologie .NET (AT.NET). Mezi tyto znalosti patřilo například práce s SMTP klientem a samotné odesílání mailů. Zároveň byla prohloubena znalost programování webových aplikací ASP.NET a znalost syntaxe jazyka C#. Co se týče syntaxe jazyka, během odborné praxe jsem se naučil standardy používané při implementaci, a to například pojmenovávaní proměnných, tříd, properties, funkcí a podobně.

Mezi další znalosti patří i poznatky, kterých se mi ve škole dostat nemůže. Jde o nový pohled na fungování společnosti v prostředí přímo mezi pracovníky, dále pak pracování na projektu ve více lidech a obecně rozšíření pohledu v oblasti chodu firmy.

# 7 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Když se ohlédnu za tím, co všechno mi odborná praxe dala, musím ji hodnotit velmi pozitivně. Nejen že jsem získal spoustu nových odborných znalostí, ale poznal jsem zblízka chod firmy a viděl, jak se pracuje na velkých projektech. V návaznosti na to hodnotím jako skvělou zkušenost setkání se s firemním produktem VENTUS, kde jsem na vlastní oči viděl, jak složitou strukturu a provázání velké informační systémy mají. Za zmínku stojí i prohloubení znalostí ve sdílení kódu a řízení vývoje pomocí nástroje TFS, se kterým jsme pracovali i jako vývojáři a své verze jsme sdíleli přes něj.

Své dosažené výsledky v rámci praxe hodnotím kladně. Všechny úkoly, které mi byly zadány jsem většinou bez velkých problémů zvládl a nakonec jsem se vždy dopracoval k správné funkcionalitě. Trochu mě mrzí, že z časových důvodů jsme nestihli systém dopracovat až k stavu, kdy by byl schopen nasazení, ale věřím, že někdo námi začatou práci dokončí a tento pomocník bude fungovat k všeobecnému prospěchu firmy.

# 8 Reference

[1] Visual Studio, *Team Foundation Server* [Online] Získáno 4 2014 z www.visualstudio.com/en-us/products/tfs-overview-vs.aspx.

[2] Ing. Marek Běhálek, Ph.D., *ADO.NET* [Online] Získáno 4 2014 z www.cs.vsb.cz/behalek/vyuka/pcsharp/text/ch07s01.html.

[3] The Code Project, *Bulding Dynamic SQL In a Stored Procedure* [Online] Získáno 4 2014 z www.codeproject.com/Articles/20815/Building-Dynamic-SQL-In-a-Stored-Procedure.

# A Výpisy kódu

```
public List<WorkItem> getItemsFromProject(string projectCode) {
    DatabaseService db = new DatabaseService(dbConnectionString);
    db.Connect();

    WorkItemFactory itemFactory = new WorkItemFactory();
    PersonService personService = new PersonService(dbConnectionString);
    ProjectService projectService = new ProjectService(dbConnectionString);

    List<WorkItem> list = new List<WorkItem>();
    SqlCommand cmd = db.CreateCommand("TasksInRequest");
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add("@requestCode", SqlDbType.VarChar, 30);
    cmd.Parameters["@requestCode"].Value = projectCode;

    SqlDataReader reader = db.Select(cmd);

    while (reader.Read()){
        WorkItem workItem = itemFactory.createTask(reader, this.getItemType((string)reader["
            UKOL_TYP_KOD"]),
            this.getItemCategory((string)reader["KATEGORIE_KOD"]), personService.
                getPersonByCode((string)reader["ZADAV_OSOBA_KOD"]),
            personService.getPersonByCode((string)reader["ZODP_OSOBA_KOD"]));
        list .Add(workItem);
    }

    reader.Close();
    db.Close();
    return list ;
}
```

Výpis 1: Funkce vracející generický list "work itemů"vyselektovaných dle rodičovského projektu

```
public bool updateWorkItem(WorkItem item){
    var db = new DatabaseService(dbConnectionString);
    db.Connect();

    var cmd = db.CreateCommand("UpdateWorkItem");
     cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add("@itemCode", SqlDbType.NVarChar, 30);
    cmd.Parameters["@itemCode"].Value = item.Code;
    cmd.Parameters.Add("@title", SqlDbType.NVarChar, 255);
     cmd.Parameters["@title"].Value = item.Name;
    cmd.Parameters.Add("@description", SqlDbType.Text);
     cmd.Parameters["@description"].Value = item.Description;
    cmd.Parameters.Add("@type", SqlDbType.NVarChar, 50);
     cmd.Parameters["@type"].Value = item.Type.Code;
    cmd.Parameters.Add("@state", SqlDbType.Int);
     cmd.Parameters["@state"].Value = item.State;
    cmd.Parameters.Add("@created", SqlDbType.DateTime);
     cmd.Parameters["@created"].Value = item.Created;
    cmd.Parameters.Add("@author", SqlDbType.NVarChar, 80);
     cmd.Parameters["@author"].Value = item.CreatedBy.Code;
    cmd.Parameters.Add("@required_date", SqlDbType.DateTime);
    cmd.Parameters["@required_date"].Value = item.CompleteDateRequired;
    cmd.Parameters.Add("@resolved_date", SqlDbType.DateTime);
     cmd.Parameters["@resolved_date"].Value = item.CompleteDate;
    cmd.Parameters.Add("@remaining_work", SqlDbType.Int);
     cmd.Parameters["@remaining_work"].Value = item.TimeLeft;
    cmd.Parameters.Add("@user", SqlDbType.VarChar, 20);
     cmd.Parameters["@user"].Value = item.AssignedTo.Code;
    cmd.Parameters.Add("@name", SqlDbType.NVarChar, 50);
     cmd.Parameters["@name"].Value = item.AssignedTo.FirstName;
    cmd.Parameters.Add("@surname", SqlDbType.NVarChar, 50);
     cmd.Parameters["@surname"].Value = item.AssignedTo.LastName;

    var retval = cmd.Parameters.Add("@body", SqlDbType.Int);
     retval.Direction = ParameterDirection.Output;
     retval.Size = 100;

     db.ExecuteNonQuery(cmd);
     int returnvalue = (int)cmd.Parameters["@body"].Value;
     db.Close();
    return Convert.ToBoolean(returnvalue);
}
```

Výpis 2: Funkce pro aktualizaci záznamů v kooperační databázi

```
create procedure UpdateServiceProcedure
(
    @state int,
    @requestCode varchar(30),
    @personCode varchar(20),
    @creatorCode varchar(20),
    @from datetime
)
as
declare @sqlRequest nvarchar(1000)
declare @tmpResult nvarchar(1000)
declare @parameterStatement nvarchar(200)
begin

set  @sqlRequest = 'select UKOL_KOD, POZAD_ZADANI_POD_KOD, NAZEV, ZADANI,
       ZADAV_OSOBA_KOD, TERMIN_ZADANI, ZODP_OSOBA_KOD, TERMIN_SPLNENI_SKUT,
       STAV, UKOL_TYP_KOD,
TERMIN_ZAHAJENI_POZAD, TERMIN_ZAHAJENI_SKUT, TERMIN_SPLNENI_POZAD,
       KATEGORIE_KOD, NAROCNOST_MNOZ_ZBYVA, NAROCNOST_MNOZ,
TERMIN_PREDANI_POZAD, TERMIN_PREDANI_SKUT, PRIPOMENUTI_OSOBA_KOD,
       TERMIN_PRIPOMENUTI, HODIN_PLAN_ZBYVA
from K_UKOLY where (1=1)'

if (@state <> 0)
    set  @sqlRequest = @sqlRequest + ' and (STAV=@state)'

if (@requestCode <> '')
    set  @sqlRequest = @sqlRequest + ' and (POZAD_ZADANI_POD_KOD=@requestCode)'

if (@personCode <> '')
    set  @sqlRequest = @sqlRequest + ' and (ZODP_OSOBA_KOD=@personCode)'

if (@creatorCode <> '')
    set  @sqlRequest = @sqlRequest + ' and (ZADAV_OSOBA_KOD=@creatorCode)'

if (@from <> '1753-1-1')
    set  @sqlRequest = @sqlRequest + ' and (TS>=@from)'

set  @parameterStatement = '@state int,
                            @requestCode varchar(30),
                            @personCode varchar(20),
                            @creatorCode varchar(20),
                            @from datetime'
exec sp_executesql @sqlRequest,
          @parameterStatement,
          @state,
          @requestCode,
          @personCode,
          @creatorCode,
          @from
end
```

Výpis 3: Dynamická procedura filtrující "work itemy"dle zadaných parametrů

```
public static void synchronizationErrorMail(string body) {

    ConfigHelper configHelper = new ConfigHelper(PathFinder.resolveRelativePath("Config.xml"));
    var server = configHelper.getSmtpServerFromXml()[0];

    var message = new MailMessage();
    message.From = new MailAddress(server.From);

    foreach (var mail in configHelper.getMailsFromXml())
        message.To.Add(new MailAddress(mail.MailAddress));

    message.Subject = "Chyba synchronizace!";
    message.Body = body;


    SmtpClient client = new SmtpClient();
     client.Host = server.Host;
     client.EnableSsl = Convert.ToBoolean(server.EnableSsl);

     if (!( server.UserName == "" && server.Password == ""))
         client.Credentials = new NetworkCredential(server.UserName, server.Password);

    try
    {
        client.Send(message);
    }
    catch (SmtpException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Výpis 4: Statická funkce pro zasílaní chybových mailů při selhání synchronizace